Apple Assembly Line

$1.50

Volume 2 -- Issue 6 March, 1982

In This Issue...

Reading Two Paddles at the Same Time.......Bob Sander-Cederlof

You may have discovered by now that if you try to read both game paddles from BASIC, there is some interaction at certain ranges. The problem is that there is only one trigger for both (really, all four) analog ports. If one of them times out long enough before the other one, you will read the tail end of the count on the second timer.

I wrote a little subroutine (see back page) which reads both paddles at once, eliminating all interaction. It also stretches the range, meaning you need a higher resistance than the standard paddles to get a full 0-255 counting range. Programs which use both paddles will run faster using this subroutine, because you get two readings in the time of one.

Circulation and Advertising Rates

Now that circulation is over 1000 copies per month, it seems appropriate to charge more per page for advertising than I did when there were only 100 to 200 subscribers. The new rate, effective immediately, is $30 for a full page, $15 for a half page.

S-C Macro Assembler........................Bob Sander-Cederlof

The printer has delivered the manuals (five days early!), the bugs are exterminated, the UPS driver went back to the depot and got a bigger truck, and we are now shipping S-C Macro Assembler.

Here is a brief summary of the new features the S-C Macro Assembler has that S-C Assembler II Version 4.0 did not. The highlights are of course macros, conditional assembly and the new commands EDIT, COPY and REPLACE. But they are not all!

Commands

There are 10 new commands:

EDIT Select a line, a range of lines, or a range of lines that contain a particular string. Edit the lines using some of the 15 convenient sub-commands.

TEXT Write source program to disk, as a TEXT file, with or without line numbers.

REPLACE Global search and replace. Your search string can include wildcards; you can limit the search to a line, a range of lines, or search the entire program. The search can be made sensitive or insensitive to upper/lower case distinctions. And you can select Auto or Verify mode for replacement.

COPY Copy one or more lines from one place to another in the source code. Rearrange your code as you please!

AUTO Generate automatic line numbers after every carriage return. Allows ordinary TEXT files to be EXECed into S-C Macro Assembler! You still can use the Version 4.0 form of automatic line numbers. Now you have a choice!

MANUAL Turn off automatic line numbering.

SYMBOLS Print out the symbol table, in case you missed it the first time.

MNTR Enter the system monitor (just like CALL -151 in BASIC). Of course all the Monitor commands can be executed within S-C Macro Assembler, but if you really WANT to leave....

RST Change the Autostart Monitor RESET vector to the specified address.

" Send setup control strings to your printer.

There are also improvements in some of the older commands.

The spelling of commands is now checked. In older versions, only the first three characters were tested. The first three are still all that are necessary, but any additional letters you type must be correct. For example, LIS will list your program, and so will LIST. But, LISX will give a syntax error.

LIST and FIND now have the same syntax (in fact, they are processed by the same routine.) They may now specify either a line range, a search string, or both. The search string now requires a delimiter.

Line ranges in the LIST, FIND, COPY, EDIT, and DELETE commands may be written with a leading or trailing comma (as in Applesoft):

LIST ,2500	List from beginning through 2500.
LIST 2500,	List from 2500 through end.

The NEW command now restarts the automatic line numbering at 1000, rather than continuing from the last line number you entered.

The SLOW and FAST commands no longer use the Monitor output hooks at $36 and $37.

To leave the Macro Assembler, type FP or INT. You no longer have to also type PR#0.

After using the PR#slot command to run your printer, use PR#0 to turn it off. FAST won't do it anymore.

Directives

There are 7 new directives:

.MA and .EM	For macro definition.
.DO expr	Start conditional block.
.ELSE	Toggle condition flag.
.FIN	End conditional block.
.TI num,title	Title and number each page of the assembly listing.
.AT string	Like .AS, but the last character has the high-bit set opposite from the rest.

The .DA directive may now have a list of expressions.

The .EQ directive may now be used with local labels.

The .LIST directive has new options to control listing of macro expansions.

Source Entry

Control-O (Override) will allow any control character to be typed into a source line in the normal input mode or in edit mode. The control character will appear in inverse video.

The editor no longer double spaces after each line is entered.

The escape-L comment line produces one less dash, so that the line lists on the screen without a blank line after it.

Operand expressions can now include * and / as operators, as well as + and -. The relational operators (<, =, and >) may also be used.

The tab routine has been changed to include up to five tab stops. The stop values are kept in a user-modifiable list starting at $1010. These are the actual column numbers (not 3 less, as in version 4.0). You may use any values up to column 248.

The tab character (control-I, $89) is kept at $100F now, so you can change it if you like some other character better.

Any sequence of the same character repeated 4 or more times in the source code is replaced by a token $C0, the character code, and the repeat count. (multiple blanks are still replaced by a single byte between $80 and $BF.) This reduces both the memory requirements and disk file size for your source programs.

If you want to shrink your source file a little, and if you have been using the Escape-L to generate comment lines that have all those dashes in them, type "EDIT" and hold down the RETURN and REPEAT keys until the entire program has been scanned. Type MEM before you do it, and after it is finished; you will probably notice a significant saving!

A parameter at location $1017 allows the extra compression to be turned on or off. If the contents of $1017 is $04, compression is on. If it is $FF, compression is off. You can experiment with this parameter to see what effect it has on program size.

Reference Manual

The S-C Macro Assembler comes with an all-new, 100-page manual. (At last! All the information in one place!) The manual includes chapters on source program format, commands, directives, operand expressions, macros, 6502 programming, SWEET-16, and a tutorial on using the Macro Assembler.

Assembly

Older versions of the assembler terminated assembly after finding one error. The S-C Macro Assembler keeps going, but rings the bell and prints an error message, so you know about it. If any errors are found during pass one, assembly terminates before doing pass two. At the end of assembly, the number of errors found is printed.

Typing the RETURN key during assembly will abort the assembly (even if the listing has been turned off with .LIST OFF directive).

Believe it or not, the new version assembles slightly faster than version 4.0! I measured about a 10% improvement on a large program.

All previous versions had difficulty handling forward references to variables which turned out to be in page zero. (Described on page 22 of the old blue manual.) That problem has been solved, so with S-C Macro Assembler it does not matter where you put your page-zero definitions.

Memory Usage

All page zero variables used by the assembler have been concentrated, so $00 through $1F are completely free for the user.

The standard version of the S-C Macro Assembler now occupies $1000 through $31FF. The symbol table starts at $3200 and grows upward; the source code still starts at $9600 and grows downward.

Included on the disk with the Macro Assembler is a Language Card version and a short EXEC file to load the card. This version fills the 16K RAM card from $D000 through about $F300. The symbol table begins at $1000 rather than $3200. The EXEC file configures things so that the language card contents appear to DOS as the opposite language to the one on the mother board. For example, if Applesoft is on the mother board, you type INT to get into the S-C Macro Assembler.

There are no variables within the body of the assembler. The Language Card version could be burned into ROM and placed on a firmware card, if you so desire.

Ordering

You can order the S-C Macro Assembler by phone or mail. We accept cash, checks, money orders, Visa, Mastercard, or COD. The price of the Macro Assembler is $80.00. Registered owners of S-C Assembler II Version 4.0 may upgrade to the S-C Macro Assembler for only $27.50.

EPROM Blaster Defined.......................Bob Sander-Cederlof

Several readers have asked what an EPROM blaster is. This is a device, more commonly called an EPROM programmer or writer or burner, which writes data into an EPROM. The EPSON interface has an EPROM device on it, called a "2708", which can hold 1024 bytes of data or program. (Only the first 256 bytes are actually used by EPSON.) A company called Apparat advertises a card for the Apple II which will write (burn, program, blast,...) stuff into a 2708. They call their board the "Blaster".

Mountain Computer makes the ROMWRITER board for the Apple. This board can only burn single-voltage 2716 EPROMs, the Apparat board can burn 2708s, 2716s, and 2732s, whether single or multiple voltages. And ROMWRITER costs almost twice as much.

Maybe you are asking, "What on earth is an EPROM, anyway?" EPROM stands for "Erasable Programmable Read Only Memory". The "memory" part is easy: each EPROM can hold a large number of bytes of data or program. A 2708 holds 1024 bytes, 2716 holds 2048 bytes, and a 2732 holds 4096 bytes.

"Read Only" means that once the bytes are recorded, they cannot be changed. They are permanent, even if power is removed. "Programmable" means that you and I can, with a burner or blaster", record the bytes; the chip comes un-recorded from the factory. Non-programmable ROMs are recorded during manufacturing.

"Erasable" means that you can erase what you have recorded and re-use the chip. An ultraviolet lamp is used to erase the contents; I bought a $75 EPROM Eraser from Logical Devices in Florida for the job. Maintaining the level of confusion, still other letters can be added to the acronym: EEPROMs are "Electrically" erasable; EAROMs are too (I don't know the difference between the two, if any).

Correction to Kassel's FIFO Handler.............Bill Morgan

Ever the experimenter, I started playing with Jim Kassel's FIFO Buffered Printer Handler as soon as I read about it. I learned a lot, but maybe I can spare you some difficulty with the following information.

1. Be aware that the three indices PBII, PBOI, and PBCC must be all cleared to zero before the first time you activate the handler.

2. Line 1720 was printed in AAL with a missing character. Change from

```
     1720       LDY PRINT.SLOT.SHIFTED
```

```
to   1720       LDY #PRINT.SLOT.SHIFTED
```

A Review of AMPER-MAGIC....................Bob Sander-Cederlof

AMPER-MAGIC is a utility program which makes it easy to add machine language subroutines to Applesoft programs and thereby extend the capabilities of Applesoft BASIC. It was written by Bob Nacon, one of our subscribers from New Jersey. For $75, you get a 51-page reference manual; an administrative program; and a collection of 23 subroutines, to be added to your programs.

Why We Need It

Here are some common problems that we have all had in developing machine language routines for Applesoft:

- Where do you put it? You don't want to clobber Applesoft or DOS, and you don't want either of them to clobber your routines.
- How do you get to it? CALL? Ampersand? USR?
- What do you do when you want to add a second routine?
- How do you pass data to the subroutine, and get answers back?

Most of the time we have put all of our routines at location $300-$3CF because that is a free area. It works great until you need the same space for a second or third routine. We also have been using the POKE technique of placing the machine language routine at location $300 and then calling it with CALL 768 or using the Ampersand command. This is fine for 1 or 2 routines, but you lose the full advantage of the speed of these routines waiting for them to be POKEd into memory. AMPER-MAGIC solves all of the above problems nicely.

AMPER-MAGIC hides your subroutines "underneath" your Applesoft program so that they are loaded automatically along with the Applesoft program. AMPER-MAGIC can handle 255 different subroutines of varying lengths. You can use as much space as necessary, up to the limit of memory. That solves the problem finding space for your routines.

The Ampersand ("&") command of Applesoft followed by the name of your routine is used to gain access to your subroutines. More about subroutine names later. By pointing the Ampersand vector at $3F5-$3F7 to the proper place, AMPER-MAGIC decodes the name of the routine desired and then transfers control to it. That solves the problem of linkage to more than one subroutine, and in a way that is human readable!

There is one limitation to the subroutines which can be used within AMPER-MAGIC: they must be fully relocatable subroutines. Without any change or reassembly they must be able to work at a new address.

Why? Because they are located at the end of your Applesoft program. As you edit your program, even just a little, the subroutines will probably move to a different address.

A fully relocatable subroutine is one which does not make any direct references to any address WITHIN the subroutine. There can not be any JMP, JSR, LDA, STA, etc. to an address within the subroutine. Only relative addressing branch commands may be used within subroutines.

Many of the subroutines published within AAL this past year were not fully relocatable but they could be made so easily. Maybe I will spend some time in a future issue discussing techniques on how to make subroutines fully relocatable. Roger Wagner, in his "Assembly Lines" column in Softalk Magazine, explained many of the motives and methods involved.

AMPER-MAGIC lets you select any name you wish for your subroutines, even for the subroutines in the AMPER-MAGIC library.

Names may be up to 4 bytes long. That is bytes, not necessarily characters. Applesoft tokenizes every command name or function name into a one byte token. Thus you can call your subroutines PRINT, INPUT, GET, etc. which only take up one byte each. A name like CLEAREOL is a legal AMPER-MAGIC name and only takes up 4 bytes (one for CLEAR, three for EOL). This allows you to name your own subroutines very descriptively for future reference.

To call a subroutine from within your program you simply use the & (Ampersand) followed by your subroutine name, followed by a "," and then your variables. The comma is not needed if there are no variables. For example: &GOTO,A*5 or &CLEAREOL:.

The AMPER-MAGIC administrative program is a smooth operating menu driven program which prompts you all along the way. Here is how you use it:

1. Load your Applesoft program.
2. Put the AMPER-MAGIC diskette in a drive and type EXEC AMPER-MAGIC. (Specify slot and drive if not the same as the last accessed one.)
3. Fill in information after the prompts, as required.

By following the menu and the well-written documentation, you can add, change, delete, and rename any subroutine in your program. You may add or delete any number of subroutines in one session.

You can load a subroutine directly from the keyboard in either decimal or hex. Thus many of the routines published in AAL can just be typed directly into AMPER-MAGIC.

If you have subroutines already assembled on disk, you simply tell AMPER-MAGIC the file names watch it work. AMPER-MAGIC makes room in the subroutine table at the end of your program, and loads the subroutine into your program. Really neat! Everything is handled automatically except for the subroutine name, which you must supply.

There isn't enough room here to describe all the other functions available, but suffice to say that AMPER-MAGIC gives you all the administrative functions you need to selectively add or delete any subroutines from your program easily and quickly.

Once you have finished with AMPER-MAGIC you simply EXIT via the menu. AMPER-MAGIC returns all your program pointers to their previous state, and clears itself out. Your program has now been modified and you can run it to check out the new subroutine. If you need to make further changes, just EXEC AMPER-MAGIC again.

The AMPER-MAGIC program alone is probably enough to justify its purchase, but you also get 23 ready to use subroutines. Some of these were originally published right here in AAL. Bob Nacon modified them wherever necessary to make them fully relocatable.

Here is a list of some of the subroutines on the disk:

&FIND,v$,v$,v	Find a substring in a string.
&DARY,v	Delete an array.
&GET,v,v	PEEK a two-byte value.
&GOSUB,v	GOSUB to a variable line.
&GOTO,v	GOTO to a variable line.
&INPUT,v$	Input a line containing even commas, quotation marks, or colons.

The ones listed above only give you the flavor. Remember, there are 23!

One of the best features of all of these subroutines is that all information is passed to and from the subroutines via variables, just like regular commands. No peeking or poking to set up parameters. This is a very professional touch, and makes the subroutines truly useful.

Each subroutine is described in detail, with all the information and examples you need to use them effectively.

As you can probably tell, I like this program. It provides all of us an easy way to add all those neat routines we have been working on, or wanting to work on, and never had a good way of accessing them.

AMPER-MAGIC is available from your local dealer or from AURORA Systems Inc., Madison, WI 53704.

More About the EPSON Interface................Peter Bartlett

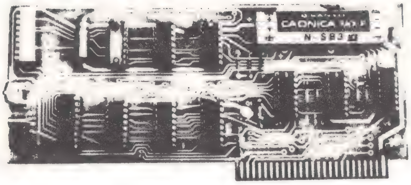
Whoops! I left out something in my instructions for modifying the EPSON interface card!

The software driver on the interface card is $100 bytes long, and resides in the first 256 bytes of the 1024-byte EPROM. However, the folks at EPSON got a couple of the address lines mixed up. Burning a new EPROM is not as straightforward as it should be.

The problem is that chunks of the program are shuffled. To understand, consider the $100 bytes to be divided into 4 parts of $40 bytes each. Part 0 is $0 to $3F, part 1 is $40 to $7F, part 2 is $80 to $BF, and part 3 is $C0 to $FF. When blasting the EPROM, the sequence of these parts must be changed. Instead of 0,1,2,3, the sequence must be 1,0,3,2.

When you list the contents of the EPROM while it is in the EPSON card, the contents appear normal. But if you remove the EPROM from the card and read it with another device, it will be in its juggled format.

Another point worth emphasizing is that this fix does not allow characters with the high-bit set to pass through the normal software driver. This driver is only compatible with the Apple's normal ASCII output. However, both Applesoft and machine language programmers can send 8-bit characters by bypassing the card as described last month in my article.



The Other EPSON Manual -- A Review.........Bob Sander-Cederlof

If you have an EPSON MX-80 printer tied to your Apple (who doesn't), you probably share the frustration of trying to learn how to use it with a manual aimed at Radio Shack TRS-80 owners. Bill Parker decided to do something about it.

Bill studied, analyzed, experimented, and perspired; then he wrote the key facts down in Apple-oriented English. With description and sample program listings he shows you how to:

1. Use all 12 print modes (emphasized, double width. etc.).
2. Underline.
3. Use subscripts and superscripts.
4. Set half-spacing, double-spacing, etc.
5. Do formfeeds, vertical tabs, etc.
6. Use horizontal tabs.
7. Use the printer commands inside a word processor.
8. Do some special tricks through the parallel interface card (true underlining, single-word emphasis, etc.).

The book(let) is 8-1/2 by 11. 17 pages, bound with a plastic comb. Not elegant. but sufficient; and anyway, it is the information he is selling. The price is $4.98, postpaid, from Bill Parker. Cut The Bull Software, P. O. Box 82761, San Diego, CA 92138.

Tricky Code that Always Skips.............Bob Sander-Cederlof

All microprocessors have an instruction which does nothing, usually called "NOP". The 6502 is no exception.

In spite of appearances, an instruction which does nothing can be quite useful. However. this article is about another kind of instruction, which does ALMOST nothing.

Some microprocessors have this kind, which do nothing except skip over one or more bytes. That is, they act like a very short forward jump. The advantage over using an actual jump or branch instruction is in memory saved. Relative branches on the 6502 take two bytes of memory; jumps take three. A skip-one or skip-two instruction would take only one byte, IF the 6502 had such.

IF? Well, you certainly do not see an instruction like this among the 56 in any of the books, do you?

However, if you disassemble things like DOS. Applesoft ROMs, and printer interface ROMs, you will find tricky ways to skip with only one byte. For example, in many Apple printer interfaces, the first three bytes look like this:

```
     C100-  18        CLC
     C101-  B0 38     BCS $C13B
```

Now isn't that silly: to clear carry, and then to use BCS to branch if it is not clear!? No, the BCS is just being used to skip over the $38 stored in $C102. If you enter the code at $C102, that $38 is a SEC instruction. Thus, depending on whether you entered at $C100 or $C102, carry is clear or set respectively. The BCS opcode byte is being used as a skip-one opcode.

Another kind of skip is found in various places inside your Apple. You might find the BIT instruction used this way. In fact, it seems to me that I run across BIT being used as a skip-one or skip-two instruction more often than I see it used to test bits! Here is an example from Applesoft ROMs:

```
     E196-  A2 78     LDX #$6B     "BAD SUBSCRIPT" MSG
     E198-  2C A2 35  BIT $35A2    TRICK: BIT SKIPS OVER 2
     E19B-  4C 12 D4  JMP $D412    PRINT ERROR MESSAGE
```

The code should really look like this:

```
     E196-  A2 78     LDX #$6B     "BAD SUBSCRIPT" MSG
     E198-  2C        .HS 2C       SKIP NEXT TWO BYTES
     E199-  A2 35     LDX #$35     "ILLEGAL QUANTITY" MSG
     E19B-  4C 12 D4  JMP $D412    PRINT ERROR MESSAGE
```

You have to be a little careful about what you skip over. The BIT instruction is actually executed, and so status flags Z, N, and V are possibly changed. Also, the two bytes skipped over represent a memory address to the BIT opcode; that memory location will be accessed. No problem, unless it just happens to be an address in the range of the I/O addresses (from $C000 to $CFFF). If it does, something strange might occur, like turning on a disk drive....

If you remember my article about the "So-Called Unused Opcodes", from about a year ago, there are some REAL skip-one and skip-two instructions. They do not modify any status bits, and they do not reference any memory addresses. I would recommend using ".HS 3C" rather than ".HS 2C" for this reason. "3C" is not a defined or supported opcode, but it apparently is built-in to all existing 6502's. (No guarantee here...test your own before you make a big commitment.)

If you want to skip only one byte, you can use the other BIT form ($24); it works on a zero page address, which will not bother any I/O addresses. If you don't want to modify any status bits, try ".HS 34".

Using the Applied Engineering Time II......Bob Sander-Cederlof

You have probably noticed Dan Pote's ad in this and previous issues of AAL. I finally got one of his clock-calendar cards, and learned how to program it.

A disk full of sample programs comes with the board, but none of them were exactly what I wanted. I wanted a simple, short program to read the time and date and display it on the screen; and I wanted some patches to DOS 3.3 which would append the date in MM/DD/YY format to any files SAVEd or BSAVEd.

The clock already had the correct time and date set when it arrived in the mail. The onboard rechargeable battery keeps the circuit running even when you remove the card from your computer! A couple of times I stopped the clock when I was working on my programs, so I just used one of the time-setting programs on the disk to correct the time.

How do you read the time and date? There are 13 registers on the board. Each register holds one digit of the time and date information. To read a particular register, you store the register number into the clock input port, and then read the clock output port.

In order to avoid reading the time or date while it is being changed, you momentarily stop the clock before reading, and restart it when you are finished. You don't want to keep the clock stopped for more than one second, or it will lose time. After stopping the clock, you have to wait at least 150 microseconds before reading it. If the clock was updating when you stopped it, the delay allows the update in progress to complete.

The following program reads the time and date and writes it on the bottom line of the screen.

```
                 1010 *-------------------------------
                 1020 *      READ DATE FROM CLOCK II
                 1030 *-------------------------------
0050-            1040 SLOT   .EQ $50      SLOT# * 16
C080-            1050 CLOCK  .EQ $C080
                 1060 *-------------------------------
0800- A2 50      1070 READ   LDX #SLOT
0802- A9 10      1080        LDA #$10     HOLD CLOCK
0804- 9D 81 C0   1090        STA CLOCK+1,X
0807- A0 00      1100        LDY #0       BEGINNING OF MAP
0809- B9 3D 08   1110 .1     LDA MAP,Y    NEXT BYTE FROM MAP
080C- F0 1F      1120        BEQ .3       END OF MAP
080E- 30 17      1130        BMI .2       COPY CHARACTER
0810- 9D 82 C0   1140        STA CLOCK+2,X    SELECT REGISTER
0813- C9 25      1150        CMP #$25     IS IT HOUR:TENS?
0815- D0 0B      1160        BNE .4       NO
0817- BD 82 C0   1170        LDA CLOCK+2,X    YES
081A- 29 03      1180        AND #3       STRIP OFF FLAGS
081C- D0 07      1190        BNE .5
081E- A9 A0      1200        LDA #$A0
0820- D0 05      1210        BNE .2       ...ALWAYS
0822- BD 82 C0   1220 .4     LDA CLOCK+2,X    READ REGISTER
0825- 09 B0      1230 .5     ORA #$B0     CONVERT TO ASCII
0827- 99 D0 07   1240 .2     STA BUFFER,Y
082A- C8         1250        INY
082B- D0 DC      1260        BNE .1       ...ALWAYS
```

```
082D- A9 00    1270 .3     LDA #0      RELEASE CLOCK
082F- 9D 81 C0 1280        STA CLOCK+1,X
0832- AD 00 C0 1290        LDA $C000    SEE IF KEY PRESSED
0835- 10 C9    1300        BPL READ     NO, KEEP READING
0837- 8D 10 C0 1310        STA $C010    YES, CLEAR STROBE
083A- 4C 8E FD 1320        JMP $FD8E    LINEFEED AND RETURN
               1330 *--------------------------------
083D- 2A 29 AF
0840- 28 27 AF
0843- 2C 2B A0
0846- A0 25 24
0849- BA 23 22
084C- BA 21 20
084F- 00       1340 MAP    .HS 2A29AF2827AF2C2BA0A02524BA2322BA212000
               1350 *--------------------------------
07D0-          1360 BUFFER .EQ $7D0
```

My Time II card is in slot 5. It will work in any slot from 1 to 7. Change line 1040 if you use a different slot. There are two addresses used to talk to the Time II card: $C081+slot*16, and $C082+slot*16. For slot 5, these are $C0D1 and $C0D2. Line 1070 loads "slot*16" into the X-register, so that loads and stores into the Time II registers will be directed to the proper slot.

Lines 1080.1090 stop the clock. Storing any value at $C0D1 of the form xxx1xxxx will stop the clock. If bit 4 is a zero the clock will be started again, as in lines 1270,1280.

Lines 1100-1260 read the date and time and store them on the screen. The reading is under the control of a format map, line 1340. The format map contains three kinds of bytes: 00, meaning the end of the map; 2x, register addresses; and ASCII characters with the high bit set. The Y-register indexes access to the map, and also the corresponding position on the screen line.

Lines 1110-1130 get the next map byte, and analyze it. If it is 00, the time and date have been read; then lines 1270-1320 restart the clock and test if you want to keep reading or not. If the byte is negative, then it is an ASCII character; Line 1240 stores the character on the screen line, and reading continues. If neither zero nor negative, the byte is a register address. Line 1140 selects the register by storing its address at $C0D2.

Lines 1150-1230 read the selected register. If the register was the tens-digit of the hour, then the flag bits are removed. These flag bits indicate whether you are using 12-hour or 24-hour format in the Time II, and AM/PM status. I didn't care, so I just mask them out. I also replaced a leading zero digit with blank here. Line 1230 converts the digit to an ASCII character.

Lines 1290-1320 test whether you have pressed any key on the keyboard. If not, reading continues. If you did, the storbe is cleared and the program terminates after printing a carriage return.

Here is a summary of the clock register addresses:

	tens	units	
Seconds	21	20	
Minutes	23	22	
Hours	25	24	with 12/24 and AM/PM flags
Day of Week		26	
Day of Month	28	27	
Month	2A	29	
Year	2C	2B	

The second program I wrote only reads the date. The actual reading is very similar to the first program, but the purpose is different. Instead of displaying it on the screen, I store in in the last 8 positions of the primary file name buffer inside DOS 3.3. The patches in lines 1040-1140 set up SAVE and BSAVE to call my program before opening the file. My program then modifies the file name to include the current date as the last 8 characters.

```
                 1010 *--------------------------------
                 1020 *      PUT DATE ON ALL NEW FILES
                 1030 *--------------------------------
                 1040        .OR $A33F    IN BSAVE COMMAND
                 1050        .TF B.1
A33F- 20 B3 B6   1060        JSR PATCH
                 1070 *--------------------------------
                 1080        .OR $A3A5    IN SAVE COMMAND
                 1090        .TF B.2
A3A5- 20 B3 B6   1100        JSR PATCH
                 1110 *--------------------------------
                 1120        .OR $A3BE    IN SAVE COMMAND
                 1130        .TF B.3
A3BE- 20 B3 B6   1140        JSR PATCH
                 1150 *--------------------------------
0005-            1160 SLOT   .EQ 5
C0D0-            1170 CLOCK  .EQ SLOT*16+$C080
                 1180 *--------------------------------
                 1190        .OR $B6B3
                 1200        .TF B.4
                 1210 PATCH
B6B3- 48         1220        PHA
B6B4- A9 10      1230        LDA #$10     HOLD CLOCK
B6B6- 8D D1 C0   1240        STA CLOCK+1
B6B9- A0 20      1250        LDY #32      DELAY 150 MICROSECONDS
B6BB- 88         1260 .1     DEY          WHILE HOLD TAKES EFFECT
B6BC- D0 FD      1270        BNE .1
B6BE- A0 07      1280        LDY #7       MOVE 8 CHARS
B6C0- B9 DC B6   1290 .2     LDA MAP,Y    NEXT BYTE FROM MAP
B6C3- 30 08      1300        BMI .3       COPY CHARACTER
B6C5- 8D D2 C0   1310        STA CLOCK+2      SELECT REGISTER
B6C8- AD D2 C0   1320        LDA CLOCK+2      READ REGISTER
B6CB- 09 B0      1330        ORA #$B0     CONVERT TO ASCII
B6CD- 99 8B AA   1340 .3     STA $AA8B,Y  IN LAST 8 CHARS OF PRIMARY FNB
B6D0- 88         1350        DEY
B6D1- 10 ED      1360        BPL .2       LOOP UNTIL ALL 8 CHARS MOVED
B6D3- A9 00      1370        LDA #0       RELEASE CLOCK
B6D5- 8D D1 C0   1380        STA CLOCK+1
B6D8- 68         1390        PLA
B6D9- 4C D5 A3   1400        JMP $A3D5    CONTINUE AFTER PATCH
                 1410 *--------------------------------
B6DC- 2A 29 AF
B6DF- 28 27 AF
B6E2- 2C 2B      1420 MAP    .HS 2A29AF2827AF2C2B
                 1430 *--------------------------------
```

I located the program inside a hole in DOS 3.3 ($B6B3-$B6FD). If you are already using a modified DOS, this hole may already have some code in it, so be careful. For example, the DOS on Applied Engineering's disk IS modified, and the modification uses this same space.

When you assemble this program, the four .TF directives write four short little binary files (B.1, B.2, B.3, and B.4). I wrote a four line EXEC file to BLOAD these four binary files, installing the patches.

The program saves the contents of the A-register at line 1220, and restores A at line 1390. Lines 1230,1240 stop the clock so we can read it. Lines 1250-1270 delay for about 150 microseconds in case the clock was updating when I stopped it.

Lines 1280-1360 read the date under control of a format map in line 1420, almost the same way the first program did. This time I used the known length of 8 bytes to terminate the loop, rather than a final 00 byte. Line 1340 stores inside the DOS primary file name buffer ($AA75-$AA92).

Lines 1370-1380 turn the clock back on. Line 1390 restores the A-register, and line 1400 continues with the normal DOS 3.3 code.

Before arriving at the above technique, I tried several others. I had one working which patched the DOS File Manager instead of the SAVE and BSAVE commands. This version appended the date to the name of any and all new files created. It worked exactly as it should, but it would have caused many problems with existing programs. Many Applesoft and Integer BASIC programs using TEXT files use an OPEN-DELETE-OPEN-WRITE sequence to make sure that a new file is used for output. If my patch to the file manager was installed, this sequence would not work correctly anymore. Therefore I elected to go the more direct route, only dating SAVE and BSAVE files.

If you want to use the date on TEXT file names, you could append it to the file name using normal string concatenation techniques.

I have not used any other of the clock/calendar cards available for the Apple, but I am convinced the Time II from Applied Engineering is a good one. (It may also be the least expensive.) The circuit card is professionally done; the components are highest quality; it works when you plug it in. There are other features, such as interrupt capability, which I have not yet explored. If you have any use for a clock/calendar, I recommend this one.

Leventhal's 6502 Subroutines

6502 Assembly Language Subroutines, by Lance Leventhal and Winthrop Saville, is a book all of you will want. Specs: 550 pages, 7-1/2 by 9-1/4 inches, paperback, $12.99 from Osborne/McGraw-Hill. I'll send you a copy for $12 plus $2 shipping (it weighs two pounds!). Naturally, shipping will be more if you live outside the USA.

Quoting from the back cover:

> "If you want to use a specific assembly language routine, learn assembly language quickly, or improve your programming skills, 6502 Assembly Language Programming is for you. It provides code for more than 40 common 6502 subroutines, including code conversion, array manipulation, arithmetic, bit manipulation, string processing, input/output, and interrupts. It describes general 6502 programming methods (including a quick summary for experienced programmers), and tells how to add instructions and addressing modes [using several instructions in sequence, subroutines, or macros]. It even discusses common 6502 assembly language programming errors."

All of the subroutines are thoroughly documented, making it easy to understand how they work, and how to use them. The subroutines are useful in the Apple with no changes, other than those required to interface to your own programs. Some of the subroutines even reference the Apple monitor ROMs!

The first five copies I bought were gone within three hours of their arrival, so I ordered 20 more. Want one?

```
:ASM

               1000 *--------------------------------
               1010 *      READ BOTH GAME PADDLES AT THE SAME TIME
               1020 *--------------------------------
0024-          1030 MON.CH .EQ $24
C064-          1040 PDL0   .EQ $C064
C065-          1050 PDL1   .EQ $C065
C070-          1060 PDL.S  .EQ $C070
C000-          1070 KEYBOARD .EQ $C000
               1080 *--------------------------------
0800- 20 1A 08 1090 TEST   JSR READ.BOTH.PADDLES
0803- 98       1100        TYA           (Y) = PDL 1 SETTING
0804- 20 DA FD 1110        JSR $FDDA     PRINT IN HEX ON SCREEN
0807- E6 24    1120        INC MON.CH    SPACE BETWEEN VALUES
0809- 8A       1130        TXA           (X) = PDL 0 SETTING
080A- 20 DA FD 1140        JSR $FDDA     PRINT IN HEX ON SCREEN
080D- A9 00    1150        LDA #0        HTAB 1
080F- 85 24    1160        STA MON.CH
0811- AD 00 C0 1170        LDA KEYBOARD SEE IF ANY KEY PRESSED
0814- 10 EA    1180        BPL TEST      NO KEYPRESS, KEEP READING PADDLES
0816- 8D 10 C0 1190        STA KEYBOARD+16   CLEAR KEYBOARD STROBE
0819- 60       1200        RTS           RETURN
               1210 *--------------------------------
               1220 READ.BOTH.PADDLES
081A- A2 00    1230        LDX #0        PADDLE 0 COUNT
081C- A0 00    1240        LDY #0        PADDLE 1 COUNT
081E- AD 70 C0 1250        LDA PDL.S     START THE PADDLE TIMERS
0821- AD 64 C0 1260 .1     LDA PDL0      CHECK PADDLE 0 TIMER
0824- 10 0D    1270        BPL .2        TIMED OUT
0826- E8       1280        INX           COUNT PDL0
0827- AD 65 C0 1290        LDA PDL1      CHECK PADDLE 1
082A- 10 17    1300        BPL .4        TIMED OUT
082C- C8       1310        INY           COUNT PDL1
082D- D0 F2    1320        BNE .1        AGAIN
082F- A2 FF    1330        LDX #255      MAX TIME FOR BOTH PADDLES
0831- D0 0C    1340        BNE .3        ...ALWAYS
               1350 *---PADDLE 0 TIMED OUT, KEEP LOOKING AT PADDLE 1
0833- AD 65 C0 1360 .2     LDA PDL1      CHECK PADDLE 1
0836- 10 19    1370        BPL .5        TIMED OUT
0838- C8       1380        INY           COUNT PDL1
0839- EA       1390        NOP           EQUALIZE TIMING
083A- EA       1400        NOP
083B- EA       1410        NOP
083C- EA       1420        NOP
083D- D0 F4    1430        BNE .2
083F- A0 FF    1440 .3     LDY #255      MAX TIME FOR PDL1
0841- D0 0E    1450        BNE .5        ...ALWAYS
               1460 *---PADDLE 1 TIMED OUT, KEEP LOOKING AT PADDLE 0
0843- AD 64 C0 1470 .4     LDA PDL0      CHECK PADDLE 0
0846- 10 09    1480        BPL .5        TIMED OUT
0848- E8       1490        INX           COUNT PDL0
0849- EA       1500        NOP           EQUALIZE TIMING
084A- EA       1510        NOP
084B- EA       1520        NOP
084C- EA       1530        NOP
084D- D0 F4    1540        BNE .4        KEEP CHECKING
084F- A2 FF    1550        LDX #255      MAX TIME FOR PDL0
0851- 60       1560 .5     RTS           RETURN TO CALLER

0000 ERRORS IN ASSEMBLY
:
```

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; $18 per year sent First Class Mail in USA, Canada, and Mexico; $28 per year sent Air Mail to other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).